XC2000 / XE166 Family

AP16188

BMI Programming for XC2000 / XE166 Econo and Compact Line Devices

Application Note

V1.0, 2011-05

Microcontrollers

Edition 2011-05

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

XC2000 / XE166
Revision History: V1.0 2011-05

Previous Version(s):

Page	Subjects (major changes since last revision)
–	This is the first release

Trademarks

Keil™ and µVision4™ are trademarks of ARM®.

We Listen to Your Comments
Is there any information in this document that you feel is wrong, unclear or missing? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Table of Contents

1 Introduction

This application note describes the start-up configuration and BMI (Boot Mode Index) programming for the Infineon Econo Line devices (XC22xx/L/D, XC23xxL/D, XC27x3X and XE16xFL) and Compact Line devices (XC22xxU/S, XC23xxU/S, XC27x2X and XE16xFU).

There are three ways to update the start-up configuration in XC2000 / XE166 Econo and Compact Line devices:

- Hardware configuration
  - The values at the configuration pins (TRST, C0 or C1) on power-on reset.
- Software configuration
  - The value in SCU_SWRSTCON.SWCFG bitfield on software reset, if SCU_SWRSTCON.SWBOOT=1.
- BMI (Boot Mode Index) configuration
  - The values taken from specific locations in Security Page 0 of the Flash Config sector at power-on.

Boot Mode Index (BMI) is a value programmed into the dedicated locations in the Flash Config sector and will be read out on power-on if the configuration pin TRST=1.

Depending on the BMI values and the configuration pins C0 and C1, or pin C0 for Compact Line devices, the devices can be started in different modes.

BMI configuration helps alleviate the problem of limited pin availability for start-up configuration on low pin count devices, and minimizes the configuration pin requirements for the various start-up modes.

2 Start-up Modes

The supported start-up modes for the XC2000 / XE166 Econo and Compact Line devices are:

- ASC Bootstrap Loader
- CAN Bootstrap Loader
- OCDS Mode
- Standard start with / without debug mode

These start-up modes can be entered on power-on reset with the correct pin and BMI values configured.

2.1 ASC Bootstrap Loader

The ASC (Asynchronous/Synchronous Serial Channel) bootstrap loader, also known as the UART (Universal Asynchronous Receiver and Transmitter) bootstrap loader, transfers program code/data via channel 0 of USIC0 (U0C0) into the PSRAM.

The UART loader only supports half-duplex communication when P10.12 is used.

After entering the UART BSL mode, the device will scan the RxD line to receive a zero byte which contains:

- One start bit
- Eight 0 data bits
- One stop bit

After receiving this zero byte, U0C0 switches pin TxD to output and transmits an identification byte (D5H) back to the host.

The required configuration pattern is summarized in the following table:

Table 1 Configuration for UART Bootstrap Loader

HWCFG Configuration	Receive and Transmit Line	Transferred Data	Supported Host Speed
0xxx $xx01_B$	RxD = TxD = P10.12	32 Bytes	2.4 - 19.2 kbaud

2.2 CAN Bootstrap Loader

The CAN bootstrap loader transfers program code/data via node 0 of the MultiCAN module into the PSRAM.

Note: The CAN bootstrap loader is not available in XC2000 / XE166 Compact line devices.

Data is transferred from the external host to the devices using eight-byte data frames. The number of data frames to be received is programmable and determined by the 16-bit data message count value, DMSGC.

The communication between the device and external host is based on the following three CAN standard frames:

- Initialization frame
  - sent by the external host to the device
- Acknowledge frame
  - sent by the device to the external host
- Data frame(s)
  - sent by the external host to the device

The initialization frame is used for baud rate detection. After a successful baud rate detection is reported to the external host by sending the acknowledge frame, data is transmitted via the data frames.

The configuration pattern for the CAN bootstrap loader is summarized in the following table:

Table 2 Configuration for CAN Bootstrap Loader

HWCFG Configuration	Receive Line from Host	Transmit Line from Host	Transferred Data	Supported Host Speed
0xxx $xx00_B$	RxDC0 = P2.6	TxDC0 = P2.5	8 x n bytes[1)]	125 - 500 kbaud

1) n = DMSGC (Data Message Count) sent by the host with the Initialization frame. Allowed values are (PSRAM_size-256)/8.

2.3 On-Chip Debug Support (OCDS) Mode

The OCDS system supports a broad range of debug features including breakpoints and memory location tracing. A typical OCDS application is to debug user software in a real-time system environment.

Three debug interfaces are available in OCDS mode for data to be transferred to/from all on and off-chip memories and memory mapped control registers. The three debug interfaces are:

- JTAG Interface (IEEE 1149.1 compliant, 4 pins)
- DAP (Device Access Port) Interface (2 pins)
- SPD (Single Pin DAP) Interface (1 pin)

*Note: The pin used for DAP and SPD interface is shown in **Table 6**. The JTAG interface is shown in **Table 7**. If debugging is enabled these pins are always assigned to the debug-interface, therefore the application software must not use these pins for any user specific function.*

3 Start-up Configuration

3.1 Hardware Configuration

When the device is powered-on the condition of the $\overline{\text{TRST}}$ configuration pin is checked.

If TRST=0, the device will start from Internal Flash without debug support execution.

If TRST=1, the device will first evaluate the BMI value. Depending on the result, the start-up mode and debug configuration are determined either from the BMI value or by configuration pins. The default value for the devices fresh from production is always for the BMI invalid condition. The devices on Easykit are programmed with DAP debug mode (DAP1=P10.12 and DAP0=P2.9).

Table 3 Start-up Mode Selection in XC2000 / XE166 Econo Line Devices

$\overline{\text{TRST}}$	BMI Value		CFG-pins		Start-up Mode
			C1 P2.9	C0 P10.12	
0	Not checked		x	x	Internal Start from Flash, no debug
1	BMI valid	BMI.BLS=disable	x	x	Start-up and debug mode from BMI
		BMI.BLS=enable	x	0	Start-up and debug mode from BMI
			x	1	ASC Bootloader
	BMI invalid / not configured		x	1	ASC Bootloader
			0	0	CAN Bootloader
			1	0	Internal start from Flash, JTAG debug

Table 4 Start-up Mode Selection in XC2000 / XE166 Compact Line Devices

$\overline{\text{TRST}}$	BMI Value		CFG-pin	Start-up Mode
			C0 P10.12	
0	Not checked		x	Internal Start from Flash, no debug
1	BMI valid	BMI.BLS=disable	x	Start-up and debug mode from BMI
		BMI.BLS=enable	0	Start-up and debug mode from BMI
			1	ASC Bootloader
	BMI invalid / not configured		1	ASC Bootloader
			0	Internal start from Flash, JTAG debug

3.2 Software Configuration

As with all other XC2000 and XE166 devices, the Econo and Compact Line devices support software configuration to start in the selected mode on software reset.

To enable software configuration, the user software must perform the following steps:

1. Install the value to select a start-up mode into SCU_SWRSTCON.SWCFG bitfield (see **Table 5** for supported values)
2. Select software boot configuration by installing SCU_SWRSTCON.SWBOOT=1
3. Trigger a software reset

Table 5 Start-up Modes in XC2000/XE166 Econo Line: Selection and Indication

Start-up Mode	HWCFG[1] / SWCFG / BMI0 bits							
	7	6	5	4	3	2	1	0
Selection by SWRSTCON.SWCFG or BMI0, indication in HWCFG								
CAN Bootloader, no debug	0	0	x	x	x	x	0	0
ASC Bootloader, no debug	0	0	x	x	x	x	0	1
Internal Start from Flash, no debug	0	0	x	x	x	x	1	0
CAN Bootloader, debug mode from HWCFG[5:2] [2]	0	1	**J**	**C**	**S**	**I1**	0	0
ASC Bootloader, debug mode from HWCFG[5:2] [2]	0	1	**J**	**C**	**S**	**I1**	0	1
Internal Start from Flash, debug mode from HWCFG[6:2] [2]	0	1	**J**	**C**	**S**	**I1**	1	0
Reserved	1	x	x	x	x	x	x	x
Selection SWRSTCON.SWCFG, HWCFG indicates BMI mode								
Start-up and debug mode from BMI	0	x	x	x	x	x	1	1

1) The bits marked with x (SWCFG/BMI0) are installed to 0 in SCU_STSTAT.HWCFG
2) For debug interface configurations, please refer to **Table 6** and **Table 7**

Table 6 SWRSTCON.SWCFG: DAP Debug Interface Configuration

DAP Interface Configuration		J	C	S	I1
DAP0 (Coded in C)	**DAP1 (Coded in I1)**				
P2.9	P10.12	0	0	x	0
P10.9	P10.12	0	1	x	0
Single Pin DAP - SPD at P10.12		0	x	x	1

Table 7 SWRSTCON.SWCFG: JTAG Debug Interface Configuration

JTAG Interface Configuration				J	C	S	I1
TCK (Coded in C)	**TMS (Coded in S)**	**TDI (Coded in I1)**	**TDO**				
P2.9	P5.4	P5.2	P10.12	1	0	0	0
P10.9	P10.11	P10.10	P10.12	1	1	1	1

3.3 Boot Mode Index (BMI) Configuration

Boot Mode Index (BMI) consists of two 16-bit words named BMI0 and BMI1, stored one after another in Flash Security Page 0, at offset 020_H...023_H.

Users need to program the BMI0 and BMI1 to enter the required start-up mode and debug configuration.

Refer to **Table 5** for the values to be programmed into BMI0 for the appropriate modes.

BMI1 is introduced to check for data consistency and is the inverse of BMI0. If BMI1 is not inverse to BMI0 (BM1 <> BMI0), BMI is considered as invalid.

3.3.1 BMI Data Structure

The BMI setting in the Config sector (BMI0/BMI1), is copied into the R0/R1 registers from Local Bank 1 at reset, to allow the user to check the values programmed.

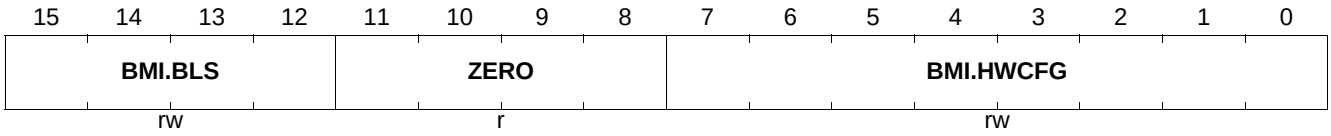
The BMI value is treated as valid if all of the following are true:

- BMI0[7:0] value selects a valid start-up mode according to **Table 5**
- BMI0[11:8] is all zero
- BMI0[15:12] = 1010B or 0101B
- BMI1 = $\overline{BMI0}$

BMI10 Register

All of the information required to select a start-up mode and a debug configuration is stored in the BMI word 0, as defined in the following register description.

BMI0
Boot Mode Index - word 0 **Flash Security Page 0** **Offset address 20_H**



Field	Bits	Type	Description
BMI.HWCFG	7:0	rw	**Start-up Mode and Debug Interface Configuration** Valid value according to **Table 5**, installed by start-up procedure into SCU_STSTAT.HWCFG
ZERO	11:8	r	**All Zero**
BMI.BLS	15:12	rw	**UART Bootstrap Loader Selection by Configuration Pin C0 (P10.12)** 1010_B**disabled** C0 is not evaluated, mode from BMI is always entered 0101_B**enabled** UART BSL mode is entered if C0=1, otherwise mode from BMI

4 BMI Installation

The Boot Mode Index (BMI) value must be programmed into Block 2 of Security page 0 in the Flash Config sector. Storing the BMI in the Config sector is installed in a different way compare to Flash Programming. Therefore, reduces the possibility of changing or destroying the BMI unintentionally.

Note: Due to the low Config sector endurance allowing a limited number of programming cycles (minimum of 10 cycles for the retention time of 20 years, according to the Data Sheet), only a limited number of BMI-changes are possible. It should be kept in mind that the endurance is defined for the Config sector as a whole; i.e. password changes are included into the number of programming cycles.

The BMI installation is intended to be carried out by third party tools, but for the advanced user who would like to implement BMI programming and verification, an example flow of the procedure is given in the User Manual, while the example code is listed in **Chapter 4.2** of this application note. The free Infineon **MemTool** (version 4.3.1 and above) supports BMI programming for Econo and Compact Line devices. **MemTool is available for download from the Infineon website**.

4.1 Installing BMI Using the Free Infineon MemTool

MemTool is a free Flash programming tool for Infineon 16-bit microcontrollers.

The BMI installation procedure is as follows:

- Start MemTool and connect the device.
- After connection between MemTool and the device is established, click the **Setup** button (**Figure 1**).

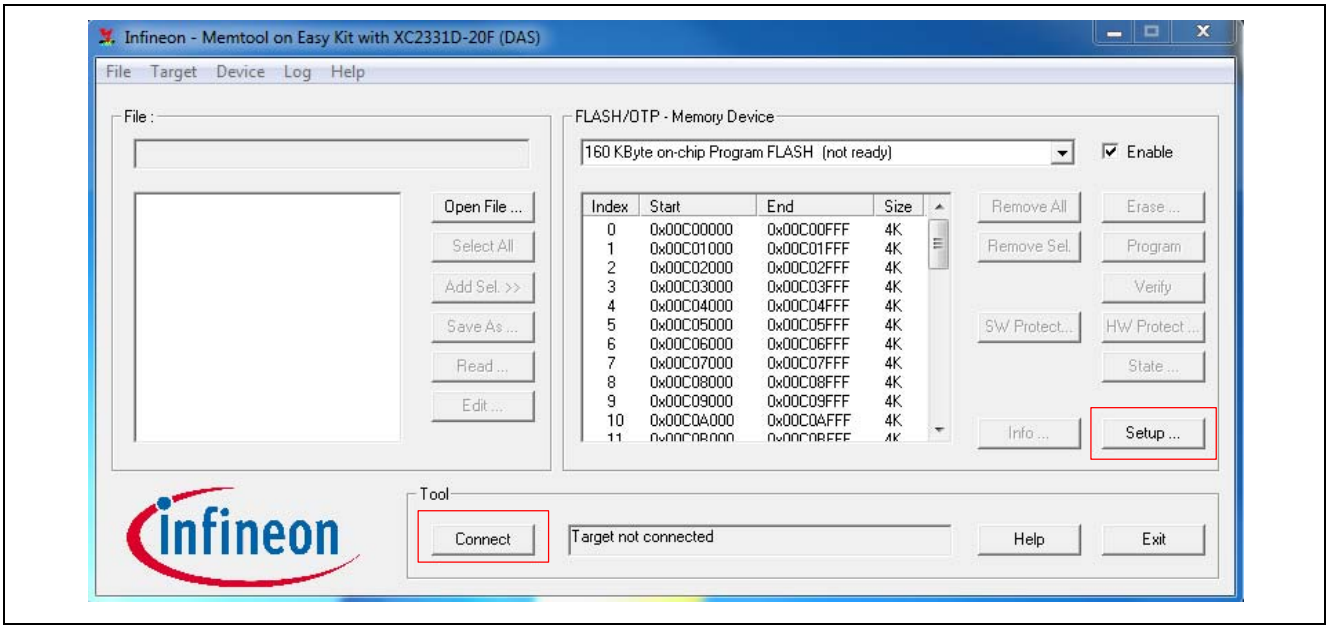


Figure 1 Infineon MemTool

- In the **Setup** window, select the **Protection/BMI** tab and check the **Install BMI configuration** box (**Figure 2**)
- Select the required start-up configuration from the drop down list.

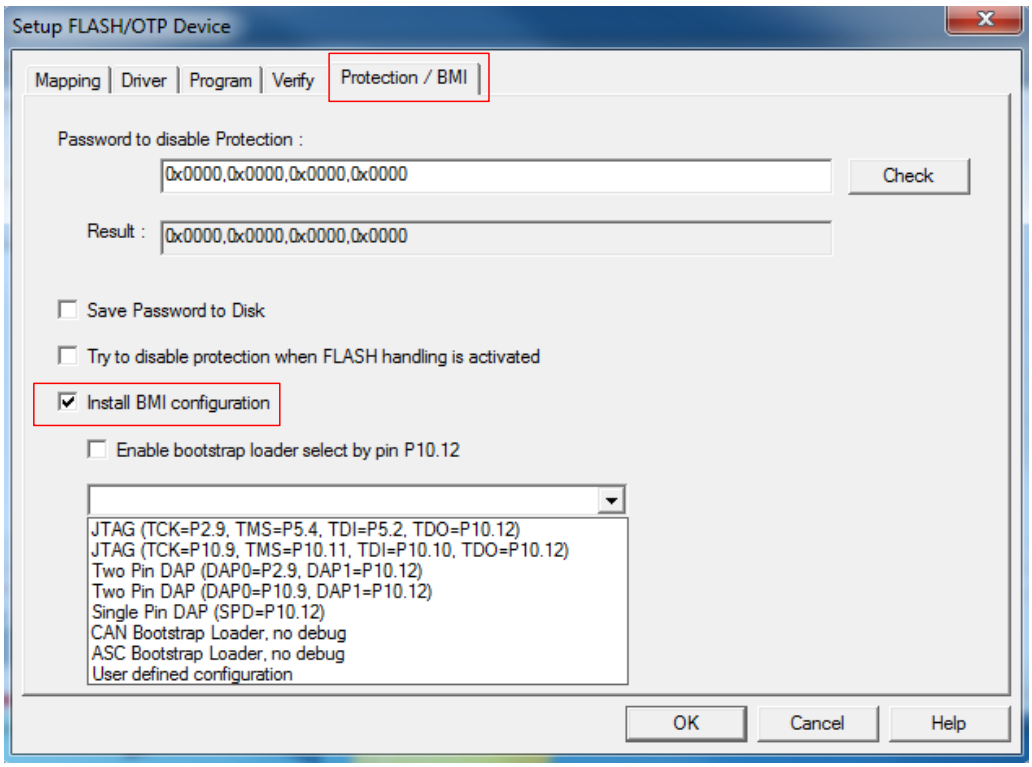


Figure 2 Setup Window in MemTool

- After selecting a start-up configuration, return to the main MemTool window and click on **HW-Protect** to open the **Enable Chip/Sector Protection** window.
- Select the **Change of current Protection** and **BMI configuration**
- Select **Start** to program the BMI value into the device.
- Once the BMI value is installed, disconnect the device and reset the target.

Attention: Before attempting to program the BMI, ensure that no sector is selected for global read/write protection if not required. This is because when the flash is protected, no debug is allowed. Device access is only possible with the Bootstrap Loader select enabled. See Chapter 4.3.2 for more details.

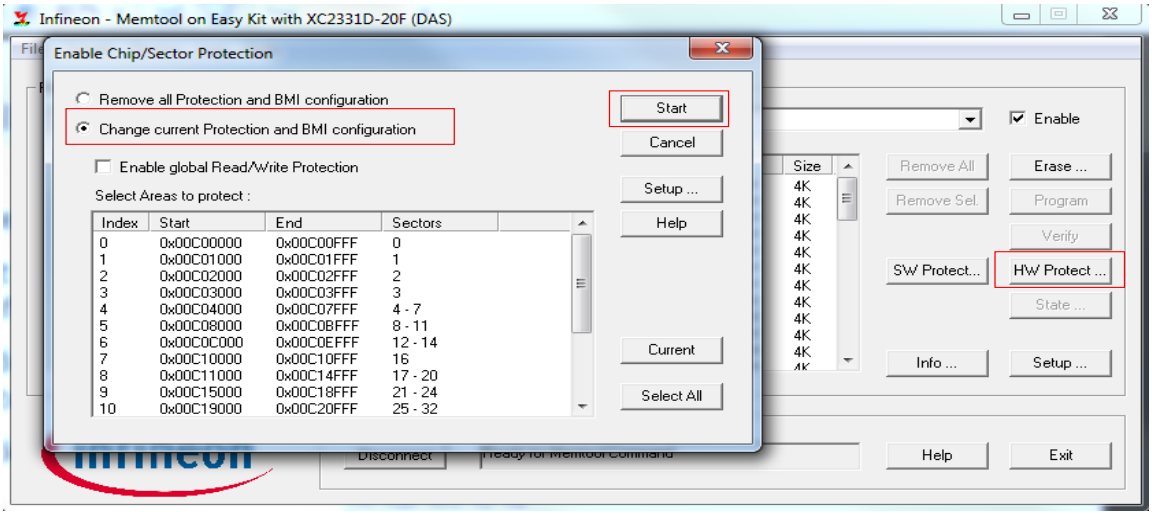


Figure 3 Program BMI Value into Device

4.2 Application Code for BMI Installation

The Boot Mode Index (BMI) can be installed via application code. Example code for BMI installation was developed in Keil™ uVision 4™, by adding additional Flash commands into the Keil™ Flash driver (FlashDev.c). This was then tested on the XE161FL-20F device. This example code can be used to determine the currently installed BMI and to make an update of the BMI value.

On reset, the start-up software will copy the current BMI value (BMI0 and BMI1) into the registers R0 and R1 of the local bank 1. This value should be copied out before the start-up code is running.

In the following example, the BMI values from register R0 and R1 is copied to the DSRAM location 0xD000 and 0xD002 respectively.

Note: Ensure that the location used to stored the BMI value will not be overwritten by any data.

The following code is added into the Keil™ Startup file (Start_V3.A66) to copy the BMI value from R0 and R1 of local bank 1.

```
OR      PSW,#00200h     ; Select Local Bank 1 -> PSW.BANK=10b
MOV     SOF(0d000h),R0 ; Move R0 value to memory location 0xD000
MOV     SOF(0d002h),R1 ; Move R1 value to memory location 0xD002
AND     PSW, #0FCFFh    ; Switch back to Global Bank
```

After storing the BMI value in DSRAM, the value can be read and further evaluated in the application software to determine the current Boot Mode.

```
void BMI_GetValue (void)
{
 int *REGptr;
 int BMI0;
 int BMI1;

 REGptr = 0xD000;
 BMI0   = *REGptr;
 REGptr = 0xD002;
 BMI1   = *REGptr;

 if (BMI0 !=  (~BMI1))
 {
       printf("BMI Invalid\n");
 }
  else
  {
    printf("BMI0: 0x%x\n",BMI0);
       BMI0 &= 0x00FF;
       if (BMI0 == 0x62)  printf ("BMI in JTAG mode\n");
    else if (BMI0 == 0x42)  printf ("BMI in DAP mode\n");
    else if (BMI0 == 0x46)  printf ("BMI in SPD mode\n");
    else printf ("BMI Invalid\n");
  }
}
```

To update the BMI value in the security page, the read and write protection must be verified using the IMBCTRH.RPA register for read protection and the IMBCTRH.WPA register for write protection.

```
void BMI_CheckProtection (void)
{
  uword WRITE;
```

```
  uword READ;
  uword PROTECTED;
  uword GetData;

  WRITE = (IMB_IMBCTRH & 0x03);
  if (WRITE == 0x01)
  {
       printf ("Flash write protection is enable\n");
       PROTECTED = 1;
  }
  else printf ("Flash write protection is disable\n");

  READ = (IMB_IMBCTRH & 0x0C);
  if (READ == 0x04)
  {
       printf ("Flash read protection is enable\n");
       PROTECTED = 1;
  }
  else printf ("Flash read protection is disable\n");

  if (PROTECTED == 1)
  {
    printf ("Disable and Erase Security Page? Y/N?\n");
    GetData = U0C0_ASC_uwGetData();
    if (GetData == 'Y')
    {
         BMI_EraseSecPage();
    }
  }
}
```

If the security page is protected, disable the protection with the protection password, then proceed to erase security page 1 which contains the security lock code.

The functions shown below are added to the Keil™ flash driver (FlashDev.c) to disable the read and write protection. The pw1, pw2, pw3 and pw4 in the function represent the passwords to disable the protection.

```
int Disable_Read_protection (unsigned int pw1, unsigned int pw2,
                     unsigned int pw3, unsigned int pw4)
{
  unsigned int far *Flash_Command_1 = (unsigned int far *) 0x0c0003C;
  unsigned int far *Flash_Command_2 = (unsigned int far *) 0x0c00054;
  unsigned int far *Flash_Command_3 = (unsigned int far *) 0x0c000AA;
  unsigned int far *Flash_Command_4 = (unsigned int far *) 0x0c00054;
  unsigned int far *Flash_Command_5 = (unsigned int far *) 0x0c000AA;
  unsigned int far *Flash_Command_6 = (unsigned int far *) 0x0c0005A;

  wait_flash_busy();

  *Flash_Command_1 = 0x0000;
  *Flash_Command_2 = pw1;
  *Flash_Command_3 = pw2;
  *Flash_Command_4 = pw3;
```

```
  *Flash_Command_5 = pw4;
  *Flash_Command_6 = 0x0055;

  wait_flash_busy();

    if ((HVAR(unsigned short, FSR_PRO)) & 0x4)return (0);
    else return (1);
}

int Disable_Write_protection (unsigned int pw1, unsigned int pw2,
                     unsigned int pw3, unsigned int pw4)
{
  unsigned int far *Flash_Command_1 = (unsigned int far *) 0x0c0003C;
  unsigned int far *Flash_Command_2 = (unsigned int far *) 0x0c00054;
  unsigned int far *Flash_Command_3 = (unsigned int far *) 0x0c000AA;
  unsigned int far *Flash_Command_4 = (unsigned int far *) 0x0c00054;
  unsigned int far *Flash_Command_5 = (unsigned int far *) 0x0c000AA;
  unsigned int far *Flash_Command_6 = (unsigned int far *) 0x0c0005A;

  wait_flash_busy();

  *Flash_Command_1 = 0x0000;
  *Flash_Command_2 = pw1;
  *Flash_Command_3 = pw2;
  *Flash_Command_4 = pw3;
  *Flash_Command_5 = pw4;
  *Flash_Command_6 = 0x0005;

  wait_flash_busy();

  if ((HVAR(unsigned short, FSR_PRO)) & 0x8)return (0);
  else return (1);
}
```

If the security page is not protected, proceed to erase security page 0. This can be done with the function that follows. This function is included in the FlashDev.c file as part of the Keil™ Flash driver function.

```
int EraseSecurityPage (unsigned long adr)
{
  unsigned int far *Flash_Command_1 = (unsigned int far *) 0x0c000AA;
  unsigned int far *Flash_Command_2 = (unsigned int far *) 0x0c00054;
  unsigned int far *Flash_Command_3 = (unsigned int far *) adr;

  /*Erase Security Page*/
  *Flash_Command_1 =0x0080;
  *Flash_Command_2 =0x00A5;
  *Flash_Command_3 =0x0053;

  wait_flash_busy();

  return (Check());
}
```

Load the BMI value into a 128-byte word-aligned buffer. The BMI0 value is stored in word 16 of the buffer and the BMI1 value, which is the inverse of BMI0, is stored in word 17.

Proceed to program the value from the buffer into the security page. The unused location in the buffer should be 00H.

```
void BMI_SetValue (char command)
{

Init_Flash(0xc00000);
ResetToRead();
ClearStatus();

printf("Erase Flash security page 0\n");
EraseSecurityPage (SecP0);

  switch (command)
       {
         case CMDSETDAP1_BMI:
              printf("Program BMI to DAP Mode (DAP0=P2.9, DAP1=P10.12)\n");
              Security_Page_0[16] = 0x0 | 0x42 | 0xA000;
              Security_Page_0[17] = ~Security_Page_0[16];
              ProgramSecurityBlock (SecP0, Security_Page_0);
              printf("Program done, please reset\n");
              break;

         case CMDSETDAP2_BMI:
              printf("Program BMI to DAP Mode (DAP0=P10.9, DAP0=P10.12)\n");
              Security_Page_0[16] = 0x0 | 0x52 | 0xA000;
              Security_Page_0[17] = ~Security_Page_0[16];
              ProgramSecurityBlock (SecP0, Security_Page_0);
              printf("Program done, please reset\n");
              break;

         case CMDSETSPD_BMI:
              printf("Program BMI to SPD Mode\n");
              Security_Page_0[16] = 0x0 | 0x46 | 0xA000;
              Security_Page_0[17] = ~Security_Page_0[16];
              ProgramSecurityBlock (SecP0, Security_Page_0);
              printf("Program done, please reset\n");
              break;

         case CMDSETJTAG1_BMI:
              printf("Program BMI to JTAG Mode (TCK=P2.9, TMS=P5.4, TDI=P5.2,
              TDO=P10.12)\n");
              Security_Page_0[16] = 0x0 | 0x62 | 0xA000;
              Security_Page_0[17] = ~Security_Page_0[16];
              ProgramSecurityBlock (SecP0, Security_Page_0);
              printf("Program done, please reset\n");
         break;

         case CMDSETJTAG2_BMI:
              printf("Program BMI to JTAG Mode (TCK=P10.9, TMS=P10.11, TDI=P10.10,
```

```
            TDO=P10.12)\n");
            Security_Page_0[16] = 0x0 | 0x7E | 0xA000;
            Security_Page_0[17] = ~Security_Page_0[16];
            ProgramSecurityBlock (SecP0, Security_Page_0);
            printf("Program done, please reset\n");
          break;

          case CMDSETCAN_BMI:
            printf("Program BMI to CAN Bootstrap Loader\n");
            Security_Page_0[16] = 0x0 | 0x00 | 0xA000;
            Security_Page_0[17] = ~Security_Page_0[16];
            ProgramSecurityBlock (SecP0, Security_Page_0);
            printf("Program done, please reset\n");
          break;

          case CMDSETASC_BMI:
            printf("Program BMI to ASC Bootstrap Loader\n");
            Security_Page_0[16] = 0x0 | 0x01 | 0xA000;
            Security_Page_0[17] = ~Security_Page_0[16];
            ProgramSecurityBlock (SecP0, Security_Page_0);
            printf("Program done, please reset\n");
          break;

          default:
            printf("Invalid Selection\n");
          break;
        }
}
```

To program the security page, enter security page mode and load the 16-bit word data into the IMB Core block assembly register. It is necessary to fill the complete page of the assembly register by a sequence of 64 “Load Page Word” commands. Next, proceed to write page command sequences and the BMI value will be written into the security page.

```
int ProgramSecurityBlock (unsigned long adr, unsigned int *buf)
{
  unsigned int count;

  unsigned int far *Flash_Command_1 = (unsigned int far *) 0x0c000AA;
  unsigned int far *Flash_Command_2 = (unsigned int far *) adr;
  unsigned int far *Flash_Command_3 = (unsigned int far *) 0x0c000F2;
  unsigned int far *Flash_Command_4 = (unsigned int far *) 0x0c0005A;

  /* Enter Security Page Mode*/
  *Flash_Command_1 = 0x0055;
  *Flash_Command_2 = 0x00AA;
  wait_flash_busy();

  /* Load Page */
  for (count=0; count < 64; count++)
  {
    *Flash_Command_3 = *buf;
    buf++;
```

```
  }

  /* Write Page */
  *Flash_Command_1 = 0x00A0;
  *Flash_Command_4 = 0x00AA;

  wait_flash_busy();

  return (Check());
}
```

After programming the new BMI value into the security page, the new Boot Mode takes effect on the next power-on reset.

4.3 Application Hint

The section provides hints on BMI configuration and a guide to ensure the device can be accessed, or to block all access per user requirement.

4.3.1 Single Pin DAP (SPD) with BMI.BLS Enable

When the device's BMI is programmed to SPD debug mode, the SPD pin P10.12 will be internal pull-up on reset.

According to **Table 3** and **Table 4**, when BMI.BLS is enabled (BMI.BLS = 0101B), UART BSL mode is entered when P10.12 = 1. Therefore, with BMI.BLS enabled, the device will go into ASC bootloader mode rather than SPD debug mode. Therefore, when using SPD mode, BMI.BLS should be disabled.

4.3.2 BMI Debug Mode with Flash Protected and BMI.BLS Disable

The Flash protection feature is used to protect user code from being read from or written to. If the Flash is protected, the device is unable to go into debug mode even when the device's BMI program is set to debug mode.

When BMI.BLS is disabled (BMI.BLS = 1010B), and the flash is protected, there is no device access. This configuration should only be employed on productive devices where no further access to the device is required or when the user needs to ensure the device is fully protected.

To allow the device to be accessed in the future, BMI.BLS should be enabled. This allows the use of the Bootstrap Loader (BSL) to access and unlock the Flash protected device.

Table 8 Settings

	BMI.BLS	**Debug**	**UART BSL**
Flash Protection	Enabled	Not allowed	Allowed
Enabled	Disabled	Not allowed	Not allowed

4.3.3 Application Software for Unlocking the Device

When the device is locked, it can not be unlocked unless there is a routine to unlock the protection in the device application program. It is the responsibility of the user to handle the accessibility of any device where this option is included. The user should therefore provide some form of security check, such as a prompt for a protection password, or to erase the Flash of the protected area before executing the unlock code.

To unlock a locked device, first disable the read and write protection with the protection password and then erase security page 1 and security page 0. These contain the lock code and the password, respectively. If this operation is successful, the locked device should be unlocked. The pseudo code for unlocking the device is below shown:

```
Disable read protection with protection password;

Disable write protection with protection password;

Erase Security Page 1;

Reset To Read;

Clear Status;

Erase Security Page 0;
```

Attention: It is the users responsibility for implementing the unlock protection routine within the user code.

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG